

PATENT ABSTRACTS OF JAPAN

(11)Publication number : 2002-351672

(43)Date of publication of application : 06.12.2002

(51)Int.Cl.

G06F 9/45

(21)Application number : 2001-156968

(71)Applicant : MITSUBISHI ELECTRIC CORP

(22)Date of filing : 25.05.2001

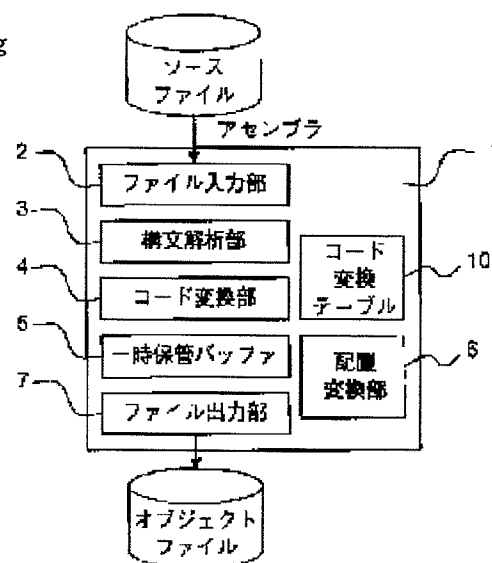
(72)Inventor : HIRATA AKIRA

(54) DEVICE AND METHOD FOR PROCESSING ASSEMBLER, AND PROGRAM FOR MAKING COMPUTER EXECUTE THE SAME METHOD

(57)Abstract:

PROBLEM TO BE SOLVED: To generate a program capable of performing the same processing even when it is operated by any of a big endian and a little endian.

SOLUTION: In this assembler processor 1 having an instruction set consisting of 16 bit length instruction and 32 bit length instruction and to generate an instruction code corresponding to a processor in which execution orders of two 16 bit length instruction codes are replaced when it is operated by a different endian, it is provided with a temporary storage buffer 5 to temporarily store the 16 bit length instruction code converted from a source file, an arrangement converting part 6 to replace the two 16 bit length instruction codes arranged in the temporary storage buffer 5 on the boundary of 32 bit and an file output part 7 to output the two replaced 16 bit length instruction codes to an object file.



(19)日本国特許庁 (J P)

(12) 公 開 特 許 公 報 (A)

(11)特許出願公開番号
特開2002-351672
(P2002-351672A)

(43)公開日 平成14年12月6日 (2002. 12. 6)

(51)Int.Cl.⁷
G 0 6 F 9/45

識別記号

F I
G 0 6 F 9/44

テ-マコ-ト*(参考)

3 2 0 D 5 B 0 8 1

審査請求 未請求 請求項の数7 O L (全 14 頁)

(21)出願番号 特願2001-156968(P2001-156968)

(22)出願日 平成13年5月25日(2001. 5. 25)

(71)出願人 000006013

三菱電機株式会社

東京都千代田区丸の内二丁目2番3号

(72)発明者 平田 明

東京都千代田区丸の内二丁目2番3号 三

菱電機株式会社内

(74)代理人 100089118

弁理士 酒井 宏明

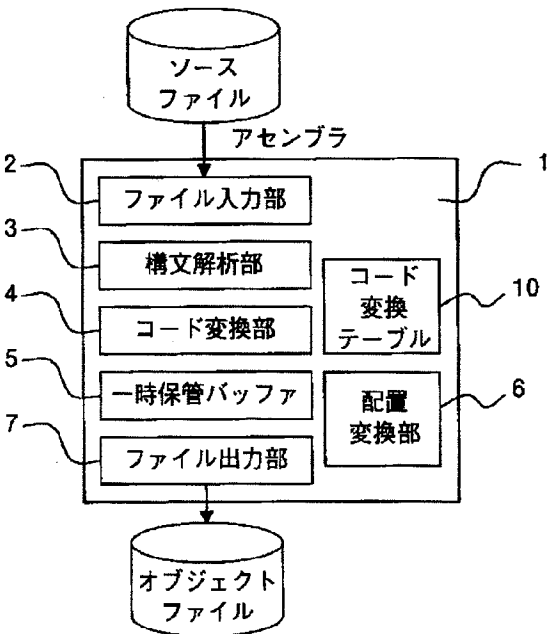
Fターム(参考) 5B081 AA07 DD01

(54)【発明の名称】 アセンブラ処理装置、アセンブラ処理方法およびその方法をコンピュータに実行させるプログラム

(57)【要約】

【課題】 ビッグエンディアン、リトルエンディアンのどちらのエンディアンで動作しても同一の処理を行えるプログラムを生成すること。

【解決手段】 16ビット長命令と32ビット長命令からなる命令セットを有するとともに異なるエンディアンで動作する時に2個の16ビット長命令コードの実行順序が入れ替わるプロセッサに対応した命令コードを生成するアセンブラ処理装置1において、ソースファイルからコード変換された16ビット長命令コードを一時的に格納する一時保管バッファ5と、一時保管バッファ5に32ビット境界で配置された2個の16ビット長命令コードを、動作対象のエンディアンに応じて入れ替える配置変換部6と、入れ替えられた2個の16ビット長命令コードを、オブジェクトファイルへ出力するファイル出力部7とを備えた。



【特許請求の範囲】

【請求項 1】 16 ビット長命令と 32 ビット長命令からなる命令セットを有するとともに異なるエンディアンで動作する時に 2 個の 16 ビット長命令コードの実行順序が入れ替わるプロセッサに対応した命令コードを生成するアセンブラ処理装置において、ソースファイルからコード変換された 16 ビット長命令コードを一時的に格納する記憶手段と、前記記憶手段に 32 ビット境界で配置された 2 個の 16 ビット長命令コードを、動作対象のエンディアンに応じて入れ替える配置変換手段と、入れ替えられた 2 個の 16 ビット長命令コードを、オブジェクトファイルへ出力するファイル出力手段と、を備えたことを特徴とするアセンブラ処理装置。

【請求項 2】 前記配置変換手段は、16 ビット長命令コードが前記記憶手段に格納されたときに、更に 16 ビット長の NOP 命令コードを前記記憶手段に追加するものであることを特徴とする請求項 1 に記載のアセンブラ処理装置。

【請求項 3】 16 ビット長命令と 32 ビット長命令からなる命令セットを有するとともに異なるエンディアンで動作する時に 2 個の 16 ビット長命令コードの実行順序が入れ替わるプロセッサに対応した命令コードを生成するアセンブラ処理装置において、2 個の 16 ビット長命令を組み合わせて 1 個の 32 ビット長命令として定義した複合命令のニーモニックと、前記複合命令の命令コードとを対応づけたコード変換テーブルと、ソースファイルから読み出した前記複合命令を、前記コード変換テーブルを参照して前記複合命令に対応する命令コードに変換するコード変換手段と、を備えたことを特徴とするアセンブラ処理装置。

【請求項 4】 16 ビット長命令と 32 ビット長命令からなる命令セットを有するとともに異なるエンディアンで動作する時に 2 個の 16 ビット長命令コードの実行順序が入れ替わるプロセッサに対応した命令コードを生成するアセンブラ処理方法において、ソースファイルからコード変換された 16 ビット長命令コードを、記憶手段に一時的に格納する記憶ステップと、前記記憶手段に 32 ビット境界で配置された 2 個の 16 ビット長命令コードを、動作対象のエンディアンに応じて入れ替える配置変換ステップと、入れ替えられた 2 個の 16 ビット長命令コードを、オブジェクトファイルへ出力するファイル出力ステップと、を含むことを特徴とするアセンブラ処理方法。

【請求項 5】 前記配置変換ステップは、16 ビット長命令コードが前記記憶手段に格納されたときに、更に 16 ビット長の NOP 命令コードを前記記憶手段に追加するものであることを特徴とする請求項 4 に記載のアセン

ブラ処理方法。

【請求項 6】 16 ビット長命令と 32 ビット長命令からなる命令セットを有するとともに異なるエンディアンで動作する時に 2 個の 16 ビット長命令コードの実行順序が入れ替わるプロセッサに対応した命令コードを生成するアセンブラ処理方法において、2 個の 16 ビット長命令を組み合わせて 1 個の 32 ビット長命令として定義した複合命令のニーモニックと前記複合命令の命令コードとを対応づけたコード変換テーブルを参照して、ソースファイルから読み出した前記複合命令を前記複合命令に対応する命令コードに変換するコード変換ステップを含むことを特徴とするアセンブラ処理方法。

【請求項 7】 請求項 4～6 のいずれか一つに記載されたアセンブラ処理方法をコンピュータに実行させるプログラム。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】この発明はソースファイルからプロセッサが実行する命令コードからなるオブジェクトファイルを生成するアセンブラ処理装置、アセンブラ処理方法およびその方法をコンピュータに実行させるプログラムに関するものであり、特にバイエンディアン・プロセッサで実行される命令コード（機械語）を生成するアセンブラ処理装置、アセンブラ処理方法およびその方法をコンピュータに実行させるプログラムに関する。

【0002】

【従来の技術】図 18 は、ソースファイルを入力しオブジェクトファイルに変換して出力する従来のアセンブラの構成を示すブロック図である。従来のアセンブラは、図 18 に示すように、ファイル入力部 12 と、構文解析部 13 と、コード変換部 14 と、オブジェクトファイル出力部 17 とを備えた構成となっている。また、図 19 は、従来のアセンブラによるアセンブラ処理のフローチャートであり、図 14 はコード変換部 14 で参照されるコード変換テーブル 10 の一部を示す説明図である。ここで、図 14 において、コード変換テーブル 10 は、アセンブリ言語のニーモニック 18 と各ニーモニックから変換される機械語（命令コード）19 とを対応させたテーブルである。

【0003】図 19 に示すように、従来のアセンブラでは、ソースファイルをファイル入力部 12 から読み込み、読み込んだソースファイルに記述された命令を構文解析部 13 によって解析する。そして、解析を行った後、コード変換部 14 がコード変換テーブル 10 を参照することによって、ニーモニックから機械語（命令コード）に変換する。変換された機械語は、ファイル出力部 17 によってファイルへ順次出力してオブジェクトファイルを生成するようになっている。

【0004】ビッグエンディアンとリトルエンディアン

とを切り替えて動作可能なプロセッサが近年登場している。ここで、ビッグエンディアンとは、データの最上位バイトがメモリ上の最下位アドレスに割り当てられるバイト順序付け方式であり、リトルエンディアンとはデータの最下位バイトがメモリ上の最下位アドレスに割り当てられるバイト順序付け方式のことをいう。プロセッサが双方のエンディアン（バイエンディアン）に対応している場合、ビッグエンディアン、リトルエンディアンのどちらで動作する場合でも、プロセッサで実行する機械語そのものはエンディアンに関係なく共通するものである。バイエンディアン対応の従来のアセンブラは、機械語をオブジェクトファイルに出力する際に機械語や扱うデータはそれぞれのエンディアンに従ったバイトの並びに変えてメモリイメージとして出力する。

【0005】図15はニーモニックを機械語に変換した時の16進数表記のコードとメモリに配置したときの状態を示す説明図である。図15では、アセンブリ言語で記述されたニーモニック20と、ニーモニック20に対応する機械語（命令コード）21と、各命令がメモリ上に配置された時のアドレス22との関係を示している。また、図15では、命令Aは32ビット長の命令、命令B、Cは16ビット長の命令を表し、命令A、B、Cの順に実行するように記述したプログラムを一例として示している。

【0006】図16と図17は、このプログラムをアセンブラ11を用いて機械語に変換したコードをオブジェクトファイルとして出力し、このオブジェクトファイルの機械語をプログラムを実行するシステムのメモリ上に展開した時のメモリイメージを示す説明図である。図16はビッグエンディアン動作を行う場合のメモリ上での命令コードの配置23を示し、図17はリトルエンディアン動作を行う場合でのメモリ上での命令コードの配置24を表している。どちらの状態においてもアセンブラで生成した命令コードは、アドレス0番地に命令A、アドレス4番地に命令B、アドレス6番地に命令Cが配置され、通常のプロセッサはアドレス順に、命令A、命令B、命令Cの順序で実行を行う。

【0007】

【発明が解決しようとする課題】ビッグエンディアンで動作し16ビットと32ビットの2種類の命令長を持つプロセッサが従来から一般的に知られている。図12はこの種のプロセッサの命令コードを示した模式図である。また、図13は、このプロセッサの動作を示すフローチャートである。図13に示す通り、プロセッサは、オブジェクトファイルから32ビット単位でデータを取り込む。そして取り込んだデータが32ビット長の命令コードである場合にはその命令コードを実行する。一方、取り込んだデータが2命令からなる16ビット長の命令コードであった場合には上位16ビットの命令コード（図12中、16bit命令1）を先に実行し、下位

16ビット（図12中、16bit命令2）の命令コードを後に実行する。また、下位16ビットの最上位ビットを同時実行フラグビットとし、このビットが1であった場合は読み込んだ2つの命令コードを並列に実行し、0であった場合には上述のように2つの16ビット長命令コードを順次実行するようになっている。

【0008】このようなビッグエンディアンで動作するプロセッサのバス・インタフェースに修正を加え、従来と異なるリトルエンディアンで動作できるようにしたプロセッサがある。このような修正を加えたプロセッサにおいても図13のフローチャートと同様に命令実行の処理を行っている。

【0009】このプロセッサがビッグエンディアンで動作した場合、図16に示したメモリイメージの命令を実行すると、0番地の32ビット[0x90F01234]を読み出し命令Aとして実行する。つぎに、4番地の32ビット[0x00C27F20]を読み出し、4番地の命令B、6番地の命令Cの順で実行する。

【0010】一方、プロセッサがリトルエンディアンで動作した場合は図17に示したメモリイメージにおいて、0番地の32ビット[0x90F01234]を読み出し、命令Aとして実行する。つぎに、4番地の32ビット[0x7F2000C2]を読み出し、6番地の命令C、4番地の命令Bの順で実行する。

【0011】このように、プロセッサでは16ビット長の2つの命令コードを順次実行した場合、ビッグエンディアンとリトルエンディアンとで命令の実行順が逆となり、リトルエンディアンの動作ではアドレス上で逆行する動作となる。

【0012】このため、リトルエンディアン動作において以上の動作を行うプロセッサで実行するプログラムを作成する場合には、2つの16ビット長命令コードを順次実行したときにアドレスと逆行することを意識したコーディングを行う必要がある。しかしながら、アセンブリ言語で記述する場合、使用する命令が32ビット長であるか16ビット長であるかを意識して記述することには困難であり、ソフトウェアの生産効率が悪くなる問題点があった。

【0013】この発明は上記に鑑みてなされたもので、どちらのエンディアンで動作しても同一の処理を行うプログラムを生成でき、アセンブリ言語プログラム開発を効率よく行って、ソフトウェアの生産効率を向上させることができるアセンブラ処理装置を得ることを目的とする。

【0014】

【課題を解決するための手段】上記目的を達成するため、この発明にかかるアセンブラ処理装置は、16ビット長命令と32ビット長命令からなる命令セットを有するとともに異なるエンディアンで動作する時に2個の1

6ビット長命令コードの実行順序が入れ替わるプロセッサに対応した命令コードを生成するアセンブラ処理装置において、ソースファイルからコード変換された16ビット長命令コードを一時的に格納する記憶手段と、前記記憶手段に32ビット境界で配置された2個の16ビット長命令コードを、動作対象のエンディアンに応じて入れ替える配置変換手段と、入れ替えられた2個の16ビット長命令コードを、オブジェクトファイルへ出力するファイル出力手段とを備えたことを特徴とする。

【0015】この発明によれば、ソースファイルからコード変換された16ビット長命令コードを記憶手段に一時的に格納し、記憶手段に32ビット境界で配置された2個の16ビット長命令コードを動作対象のエンディアンに応じて入れ替えて、2個の16ビット長命令コードを、オブジェクトファイルへ出力するので、プロセッサがビッグエンディアンとリトルエンディアンのどちらのエンディアンで動作する場合でも、アセンブリ言語記述の同一ソースファイルを用いて同一の処理を実行するオブジェクトファイルを生成することができ、ソフトウェアの生産性を改善することができる。

【0016】つぎの発明にかかるアセンブラ処理装置は、上記の発明において、前記配置変換手段は、16ビット長命令コードが前記記憶手段に格納されたときに、更に16ビット長のNOP命令コードを前記記憶手段に追加するものであることを特徴とする。

【0017】この発明によれば、16ビット長命令コードが記憶手段に格納されたときに、更に16ビット長のNOP命令コードを記憶手段に追加するので、全ての16ビット長命令を32ビットの境界に配置して、リトルエンディアン動作時において16ビット長命令がアドレスに逆行した実行動作となることを無視することができる。また、逆アセンブルでメモリモニックの対応表示を行った場合にはコーディングした手順がそのまま出力されるため、検証には都合がよくソフトウェア開発の効率化を行うことができる。

【0018】つぎの発明にかかるアセンブラ処理装置は、16ビット長命令と32ビット長命令からなる命令セットを有するとともに異なるエンディアンで動作する時に2個の16ビット長命令コードの実行順序が入れ替わるプロセッサに対応した命令コードを生成するアセンブラ処理装置において、2個の16ビット長命令を組み合わせる1個の32ビット長命令として定義した複合命令のニーモニックと、前記複合命令の命令コードとを対応づけたコード変換テーブルと、ソースファイルから読み出した前記複合命令を、前記コード変換テーブルを参照して前記複合命令に対応する命令コードに変換するコード変換手段と、を備えたことを特徴とする。

【0019】この発明によれば、2個の16ビット長命令を組み合わせる1個の32ビット長命令を複合命令として定義するとともに、この複合命令のニーモニックと

命令コードとを対応づけたコード変換テーブルを定義しておく。そして、このコード変換テーブルを用いて、ソースファイルから読み出した複合命令を複合命令の命令コードにコード変換するので、生成される命令コードはすべて32ビット長となる。このため、プロセッサのリトルエンディアン動作時における命令の逆転実行を確実になくすることができ、また、コード量の最適化を図ることができる。

【0020】つぎの発明にかかるアセンブラ処理方法は、16ビット長命令と32ビット長命令からなる命令セットを有するとともに異なるエンディアンで動作する時に2個の16ビット長命令コードの実行順序が入れ替わるプロセッサに対応した命令コードを生成するアセンブラ処理方法において、ソースファイルからコード変換された16ビット長命令コードを、記憶手段に一時的に格納する記憶ステップと、前記記憶手段に32ビット境界で配置された2個の16ビット長命令コードを、動作対象のエンディアンに応じて入れ替える配置変換ステップと、入れ替えられた2個の16ビット長命令コードを、オブジェクトファイルへ出力するファイル出力ステップとを含むことを特徴とする。

【0021】この発明によれば、ソースファイルからコード変換された16ビット長命令コードを記憶手段に一時的に格納し、記憶手段に32ビット境界で配置された2個の16ビット長命令コードを動作対象のエンディアンに応じて入れ替えて、2個の16ビット長命令コードを、オブジェクトファイルへ出力するので、プロセッサがビッグエンディアンとリトルエンディアンのどちらのエンディアンで動作する場合でも、アセンブリ言語記述の同一ソースファイルを用いて同一の処理を実行するオブジェクトファイルを生成することができ、ソフトウェアの生産性を改善することができる。

【0022】つぎの発明にかかるアセンブラ処理方法は、上記の発明において、前記配置変換ステップは、16ビット長命令コードが前記記憶手段に格納されたときに、更に16ビット長のNOP命令コードを前記記憶手段に追加するものであることを特徴とする。

【0023】この発明によれば、16ビット長命令コードが記憶手段に格納されたときに、更に16ビット長のNOP命令コードを記憶手段に追加するので、全ての16ビット長命令を32ビットの境界に配置して、リトルエンディアン動作時において16ビット長命令がアドレスに逆行した実行動作となることを無視することができる。また、逆アセンブルでメモリモニックの対応表示を行った場合にはコーディングした手順がそのまま出力されるため、検証には都合がよくソフトウェア開発の効率化を行うことができる。

【0024】つぎの発明にかかるアセンブラ処理方法は、16ビット長命令と32ビット長命令からなる命令セットを有するとともに異なるエンディアンで動作する

時に 2 個の 16 ビット長命令コードの実行順序が入れ替わるプロセッサに対応した命令コードを生成するアセンブラ処理方法において、2 個の 16 ビット長命令を組み合わせる 1 個の 32 ビット長命令として定義した複合命令のニーモニックと前記複合命令の命令コードとを対応づけたコード変換テーブルを参照して、ソースファイルから読み出した前記複合命令を前記複合命令に対応する命令コードに変換するコード変換ステップを含むことを特徴とする。

【0025】この発明によれば、2 個の 16 ビット長命令を組み合わせた 1 個の 32 ビット長命令を複合命令として定義するとともに、この複合命令のニーモニックと命令コードとを対応づけたコード変換テーブルを定義しておく。そして、このコード変換テーブルを用いて、ソースファイルから読み出した複合命令を複合命令の命令コードにコード変換するので、生成される命令コードはすべて 32 ビット長となる。このため、プロセッサのリトルエンディアン動作時における命令の逆転実行を確実になくすことができ、また、コード量の最適化を図ることができる。

【0026】つぎの発明にかかるプログラムは、上述した発明のいずれか一つに記載されたアセンブラ処理方法をコンピュータに実行させるプログラムであり、そのプログラムが機械読み取り可能となり、これによって、上記の発明のいずれか一つの動作をコンピュータによって実行することができる。

【0027】

【発明の実施の形態】以下に添付図面を参照して、この発明にかかるアセンブラ処理装置、アセンブラ処理方法およびその方法をコンピュータに実行させるプログラムの好適な実施の形態を詳細に説明する。

【0028】実施の形態 1. 図 1 は、この発明の実施の形態 1 であるアセンブラの構成を示すブロック図である。実施の形態 1 のアセンブラは、アセンブラ言語で記述されたソースファイルからビッグエンディアン用の命令コード（機械語）とリトルエンディアン用の命令コード（機械語）をオブジェクトファイルとして出力するバイエンディアン対応アセンブラである。また、実施の形態 1 のアセンブラは、上述した図 12 に示す命令コードを有し、かつ上述した図 13 のフローチャートに示す動作を行うプロセッサで実行される命令コードを生成するものである。

【0029】図 1 に示すように、実施の形態 1 のバイエンディアン対応アセンブラ 1 は、ソースファイルを読み出すファイル入力部 2 と、ファイル入力部 2 で読み出したソースファイルに記述されたアセンブリ言語のソースコードを構文解析する構文解析部 3 と、コード変換テーブル 10 と、コード変換テーブル 10 を参照して、アセンブリ言語のニーモニックを命令コード（機械語）に変換するコード変換部 4 と、変換された命令コードを一時

的に記憶する一時保管バッファ 5 と、一時保管バッファ 5 に保存された命令コードの配置を変換する配置変換部 6 と、変換された命令コードまたは一時保管バッファ 5 に記憶されている変換された命令コードを、オブジェクトファイルとして出力処理を行うファイル出力部 7 とを備えている。

【0030】コード変換テーブル 10 は、アセンブラ言語のニーモニック表記と命令コードとを対応付けた表であり、図 14 はコード変換テーブル 10 の一例を示す説明図である。一時保管バッファ 5 は、アセンブラ 1 が実行されるときに、記憶手段としてのメモリ上に 4 個のバイトデータの配列（char BUF [3]）を確保される。配置変換部 6 は、一時保管バッファ 5 に記憶された命令コードの配置を、動作するエンディアンに応じてバイト単位で変換するものである。

【0031】つぎに、以上のように構成されたバイエンディアン対応アセンブラによる、ビッグエンディアン用の命令コード生成の処理手順について説明する。図 2 は、ビッグエンディアン用命令コード生成の処理手順を示すフローチャートである。

【0032】アセンブラ 1 は、まず、一時保管バッファ 5 としてメモリ上に 4 個のバイトデータの配列（char BUF [3]）を確保して、これを初期化する（ステップ S201）。そして、ファイル入力部 2 によりアセンブリ言語で記述されたソースファイルを一行分読み込み（ステップ S202）、読み込んだソースコードを構文解析部 3 によって構文解析する（ステップ S203）。構文解析終了後、コード変換部 4 においてニーモニックと命令コード（機械語）の対応表が記されているコード変換テーブル 10（図 14）を参照してソースコードを命令コード（機械語）に変換する（ステップ S204）。

【0033】そして、配置変換部 6 では、更に以下のビッグエンディアン用の命令コード生成のための配置変換処理を行う。まず、コード変換された命令コードのサイズをチェックする（ステップ S205）。命令コードが 32 ビット長であった場合には、更に一時保管バッファ 5 に前回変換分の命令コードが残存しているか否かをチェックする（ステップ S206）。残存していなければ、変換された命令コードの上位バイトから 1 バイトずつ 4 バイト分を、一時保管バッファ 5 に BUF [0]、BUF [1]、BUF [2]、BUF [3] の順で格納する（ステップ S209）。そして、ファイル出力部 7 によって、一時保管バッファ 5 に格納された命令コードをオブジェクトファイルに出力する（ステップ S213）。出力する順番及びバイトごとの配置は、BUF [0]、BUF [1]、BUF [2]、BUF [3] の順番で、ビッグエンディアンに対応させてオブジェクトファイルの下位アドレス（即ち、MSB）から出力する。

【0034】ステップS205におけるコードサイズのチェックの結果、変換された命令コードが16ビット長であった場合は、一時保管バッファ5に前回変換分の命令コードが残存しているか否かをチェックする(ステップS210)。残存していなければ、16ビット長命令コードの上位8ビットを一時保管バッファ5のBUF [0]に、下位8ビットをBUF [1]に保管する(ステップS211)。

【0035】つぎに、ファイル入力部2においてソースファイルからつぎの一行のソースコードを読み出し(ステップS202)、同様にステップS205までの処理を行う。この命令が16ビット長命令コードであった場合、一時保管バッファ5に前回変換分の命令コードが残存しているか否かをチェックするが(ステップS210)、この場合、BUF [0]とBUF [1]に残存しているので、新たに変換した命令コードの上位8ビットをBUF [2]に、下位8ビットをBUF [3]に保存する(ステップS212)。そして、ファイル出力部7によって、一時保管バッファ5に格納された命令コードを、BUF [0]、BUF [1]、BUF [2]、BUF [3]の順番で、ビッグエンディアンに対応させてオブジェクトファイルの下位アドレス(即ち、MSB)から出力する(ステップS213)。

【0036】ステップS205で、16ビット長命令コードのつぎに読み出して変換された命令コードが32ビット長命令コードであった場合には、一時保管バッファ5に前回変換分の命令コードが残存しているか否かをチェックするが(ステップS206)、この場合、BUF [0]とBUF [1]に残存しているので、ステップS207へ進み、一時保管バッファ5のBUF [2]、BUF [3]に16ビットの無処理命令(NOP)を追加保存する。そして、ファイル出力部7によって、一時保管バッファ5に格納された命令コードを、BUF [0]、BUF [1]、BUF [2]、BUF [3]の順番で、ビッグエンディアンに対応させてオブジェクトファイルの下位アドレス(即ち、MSB)から出力し(ステップS208)、続いて次の32ビット長命令コードの一時保管バッファ5への保存(ステップS209)およびオブジェクトファイルへの出力を行う(ステップS213)。

【0037】図4(a)は、一例としてソースコードが32ビット長の命令コードA、16ビット長の命令コードB、および16ビット長の命令コードCにこの順番でコード変換された場合の、ビッグエンディアン用の各命令コードの一時保管バッファ5への格納状態、およびオブジェクトファイルへの出力状態を示す説明図である。図4(b)は、一例としてソースコードが16ビット長の命令コードB、32ビット長の命令コードAにこの順番でコード変換された場合の、各命令コードの一時保管バッファ5への格納状態、およびオブジェクトファイル

への出力状態を示す説明図である。

【0038】プロセッサがビッグエンディアンで動作する場合、プロセッサがオブジェクトファイルからメモリ上にロードされた命令コードを0番地、4番地、6番地の順に実行していくが、図4(a)および(b)からわかるように、オブジェクトファイルへの各命令コードの格納配置はビッグエンディアンの場合に命令コードの実行順に配置されるよう変換されるので、ソースファイルの命令の順番と命令の実行順が一致したものとなる。

【0039】つぎに、実施の形態1のアセンブラ1によるリトルエンディアン用の命令コード生成の処理手順について説明する。図3は、リトルエンディアン用命令コード生成の処理手順を示すフローチャートである。アセンブラ1は、ビッグエンディアン用命令コード生成処理の場合と同様に、一時保管バッファ5の初期化(ステップS301)、ソースファイルから一行読み出し(ステップS302)、構文解析(ステップS303)、コード変換を行う(ステップS304)。そして配置変換部6では、更に以下のリトルエンディアン用の命令コード生成のための配置変換処理を行う。まず、コード変換された命令コードのサイズをチェックする(ステップS305)。命令コードが32ビット長であった場合には、更に一時保管バッファ5に前回変換分の命令コードが残存しているか否かをチェックする(ステップS306)。残存していなければ、コード変換された命令コードの下位バイトから1バイトずつ4バイト分を、一時保管バッファ5にBUF [0]、BUF [1]、BUF [2]、BUF [3]の順で格納する(ステップS309)。そして、ファイル出力部7によって、一時保管バッファ5に格納された命令コードをオブジェクトファイルに出力する(ステップS313)。出力する順番及びバイトごとの配置は、BUF [0]、BUF [1]、BUF [2]、BUF [3]の順番で、リトルエンディアンに対応させてオブジェクトファイルの下位アドレス(即ち、LSB)から出力する。

【0040】ステップS305におけるコードサイズチェックの結果、変換された命令コードが16ビット長であった場合は、一時保管バッファ5に前回変換分の命令コードが残存しているか否かをチェックする(ステップS310)。残存していなければ、16ビット長命令コードの上位8ビットを一時保管バッファ5のBUF [3]に、下位8ビットをBUF [2]に保管する(ステップS311)。

【0041】つぎに、ファイル入力部2においてソースファイルからつぎの一行のソースコードを読み出し(ステップS302)、同様にステップS305までの処理を行う。この命令が16ビット長命令コードであった場合、一時保管バッファ5に前回変換分の命令コードが残存しているか否かをチェックするが(ステップS310)、この場合、BUF [3]とBUF [2]に残存し

ているので、新たに変換した命令コードの上位8ビットをBUF [1] に、下位8ビットをBUF [0] に保存する(ステップS312)。そして、ファイル出力部7によって、一時保管バッファ5に格納された命令コードを、BUF [0]、BUF [1]、BUF [2]、BUF [3] の順番で、リトルエンディアンに対応させてオブジェクトファイルの下位アドレス(即ち、LSB)から出力する(ステップS313)。

【0042】ステップS305で、16ビット長命令コードのつぎに読み出して変換された命令コードが32ビット長命令であった場合には、一時保管バッファ5に前回変換分の命令コードが残存しているか否かをチェックするが(ステップS306)、この場合、BUF [3] とBUF [2] に残存しているので、ステップS307へ進み、一時保管バッファ5のBUF [1]、BUF [0] に16ビットの無処理命令(NOP)を追加保存する。そして、ファイル出力部7によって、一時保管バッファ5に格納された命令コードを、BUF [0]、BUF [1]、BUF [2]、BUF [3] の順番で、リトルエンディアンに対応させてオブジェクトファイルの下位アドレス(即ち、LSB)から出力し(ステップS308)、続いて次の32ビット長命令コードの一時保管バッファ5への保存(ステップS309)およびオブジェクトファイルへの出力を行う(ステップS313)。

【0043】図5(a)は、一例としてソースコードが32ビット長の命令コードA、16ビット長の命令コードB、および16ビット長の命令コードCにこの順番でコード変換された場合の、リトルエンディアン用の各命令コードの一時保管バッファ5への格納状態、およびオブジェクトファイルへの出力状態を示す説明図である。図5(b)は、一例としてソースコードが16ビット長の命令コードB、32ビット長の命令コードAにこの順番でコード変換された場合の、各命令コードの一時保管バッファ5への格納状態、およびオブジェクトファイルへの出力状態を示す説明図である。

【0044】プロセッサがリトルエンディアンで動作する場合にも、プロセッサがオブジェクトファイルからメモリ上にロードされた命令コードを0番地、4番地、6番地の順に実行していくが、図5(a)および(b)からわかるように、オブジェクトファイルへの各命令コードの格納配置はリトルエンディアンの場合に命令コードの実行順に配置されるよう変換されるので、ソースファイルの命令の順番と命令の実行順が一致したものとなる。

【0045】以上説明したようにアセンブラ処理を行うことでアセンブリ言語記述の同一ソースファイルを用いてもプロセッサにおいてどちらのエンディアンにおいても同一の処理を実行するオブジェクトファイルを生成することができ、ソフトウェアの生産性を改善することが

できる。

【0046】実施の形態2. 実施の形態1のバイエンディアン対応アセンブラは、32ビット境界で16ビット長命令に続く命令が16ビット長命令であった場合は2つの命令を繋げて32ビット分を変換した後にオブジェクトファイルに出力するものであったが、この実施の形態2のバイエンディアン対応アセンブラは、16ビット命令を変換する際に残り半分の16ビットを常にNOP命令とすることで処理の簡略化を行うものである。

【0047】図6は、この発明の実施の形態2であるアセンブラの構成を示すブロック図である。図6に示すように、実施の形態2のバイエンディアン対応アセンブラは、ファイル入力部2と、構文解析部3と、コード変換部4と、コード変換テーブル10と、一時保管バッファ5と、配置変換部6と、ファイル出力部7とを備えている。ここで、ファイル入力部2と、構文解析部3と、コード変換部4と、コード変換テーブル10と、一時保管バッファ5と、ファイル出力部7は、実施の形態1のアセンブラと同様であるため、説明を省略する。配置変換部6は、一時保管バッファ5に16ビット長命令コードが保存された場合に、NOP命令を追加して動作するエンディアンに応じて16ビット長命令コードとNOP命令の配置を変換するものである。

【0048】つぎに、このように構成された実施の形態2のバイエンディアン対応アセンブラによるビッグエンディアン用の命令コード生成処理手順について説明する。図7は、ビッグエンディアン用命令コード生成の処理手順を示すフローチャートである。

【0049】アセンブラ1は、実施の形態1のビッグエンディアン用命令コード生成処理と同様に、一時保管バッファ5の初期化(ステップS601)、ソースファイルから一行読み出し(ステップS602)、構文解析(ステップS603)、コード変換を行う(ステップS604)。そして配置変換部6では、更に以下のリトルエンディアン用の命令コード生成のための配置変換処理を行う。まず、コード変換された命令コードのサイズをチェックする(ステップS605)。

【0050】命令コードが32ビット長であった場合には、コード変換された命令コードの上位バイトから1バイトずつ4バイト分を、一時保管バッファ5にBUF [0]、BUF [1]、BUF [2]、BUF [3] の順で格納する(ステップS606)。そして、ファイル出力部7によって、一時保管バッファ5に格納された命令コードを、BUF [0]、BUF [1]、BUF [2]、BUF [3] の順番で、ビッグエンディアンに対応させてオブジェクトファイルの下位アドレス(即ち、MSB)から出力する(ステップS610)。

【0051】ステップS605におけるコードサイズのチェックの結果、変換された命令コードが16ビット長であった場合は、16ビット長命令コードの上位8ビッ

トを一時保管バッファ5のBUF [0] に、下位8ビットをBUF [1] に保管する(ステップS607)。そして、一時保管バッファ5のBUF [2]、BUF [3] に16ビットの無処理命令(NOP)を追加保存する(ステップS608)。更に、NOP命令を保存するバッファのBUF [2] の最上位ビットに「1」を設定する(ステップS609)。BUF [2] の最上位ビットは、図13の動作を行うプロセッサが命令コードを実行するときに、読み込んだ2つの16ビット長の命令を並列に実行するか否かを判断する際の並列実行ビットに対応するものである。かかる最上位ビットを「1」に設定することにより、BUF [0] とBUF [1] に保存された16ビット長命令とBUF [2] とBUF [3] に保存されたNOP命令とがプロセッサによって並列に実行されることになる。

【0052】そして、ファイル出力部7によって、一時保管バッファ5に格納された命令コードを、ビッグエンディアンに対応するため、BUF [0]、BUF [1]、BUF [2]、BUF [3] の順番で、ビッグエンディアンに対応させてオブジェクトファイルの下位アドレス(即ち、MSB)から出力する(ステップS610)。

【0053】つぎに、バイエンディアン対応アセンブラによるリトルエンディアン用の命令コード生成処理手順について説明する。図8は、リトルエンディアン用命令コード生成の処理手順を示すフローチャートである。

【0054】アセンブラ1は、上述のビッグエンディアン用命令コード生成処理と同様に、一時保管バッファ5の初期化(ステップS701)、ソースファイルから一行読出し(ステップS702)、構文解析(ステップS703)、コード変換を行う(ステップS704)。そして配置変換部66では、更に以下のリトルエンディアン用の命令コード生成のための配置変換処理を行う。

【0055】まず、コード変換された命令コードのサイズをチェックする(ステップS705)。命令コードが32ビット長であった場合には、コード変換された命令コードの下位バイトから1バイトずつ4バイト分を、一時保管バッファ5にBUF [0]、BUF [1]、BUF [2]、BUF [3] の順で格納する(ステップS706)。そして、ファイル出力部7によって、BUF [0]、BUF [1]、BUF [2]、BUF [3] の順番で、リトルエンディアンに対応させてオブジェクトファイルの下位アドレス(即ち、LSB)から一時保管バッファ5に格納された命令コードをオブジェクトファイルに出力する(ステップS710)。

【0056】ステップS705におけるコードサイズのチェックの結果、コード変換された命令コードが16ビット長であった場合は、16ビット長命令コードの上位8ビットを一時保管バッファ5のBUF [3] に、下位8ビットをBUF [2] に保管する。(ステップS70

7)。そして、一時保管バッファ5のBUF [1]、BUF [0] に16ビットの無処理命令(NOP)を追加保存する(ステップS708)。更に、16ビット長命令コードを保存するバッファのBUF [2] の最上位ビットに「1」を設定する(ステップS709)。このBUF [2] の最上位ビットは、ビッグエンディアン用の処理と同様に、並列実行ビットに対応するものである。かかる最上位ビットを「1」に設定することにより、BUF [3] とBUF [2] に保存された16ビット長命令とBUF [1] とBUF [0] に保存されたNOP命令とがプロセッサによって並列に実行されることになる。

【0057】そして、ファイル出力部7によって、一時保管バッファ5に格納された命令コードを、BUF [0]、BUF [1]、BUF [2]、BUF [3] の順番で、リトルエンディアンに対応させてオブジェクトファイルの下位アドレス(即ち、LSB)から出力する(ステップS710)。

【0058】このように、実施の形態2のアセンブラでは、全ての16ビット長命令を32ビットの境界に配置するようにしているので、リトルエンディアン動作時において16ビット長命令がアドレスに逆行した実行動作となることを無視することができ、また逆アセンブルでメモリーイメージとニーモニックの対応表示を行った場合にはコーディングした手順がそのまま出力されるため検証には都合がよくソフトウェア開発の効率化を行うことができる。また、並列実行ビットに対応するビットを「1」に設定しているので、16ビット長命令と16ビット長のNOP命令とを並列実行させることができ、NOP命令を追加したことによるプロセッサの実行時間の増加を少なくすることができる。

【0059】実施の形態3. 実施の形態1および実施の形態2のバイエンディアン対応アセンブラは、ビッグエンディアンとリトルエンディアンの動作モードにおいて同一のソースファイルを用いることができる利点があるが、リトルエンディアンで動作したときのプロセッサでは命令実行におけるアドレスの逆行現象は避けられない。そこで、実施の形態3のバイエンディアン対応アセンブラでは、16ビット長命令の全ての組み合わせについて複合命令のニーモニックを定義し、この複合命令で記述されたソースファイルを命令コードに変換することによりアドレスの逆行を完全に防止するものである。

【0060】図9は、実施の形態3のアセンブラの構成を示すブロック図である。実施の形態3のアセンブラ1は、図9に示すように、ファイル入力部2と、構文解析部3と、コード変換部904と、コード変換テーブル910と、ファイル出力部7とを備えている。ここで、ファイル入力部2と、構文解析部3と、ファイル出力部7は、従来のアセンブラと同様の構成であるため説明を省略する。

【0061】図10は、実施の形態3のアセンブラで用いられる複合命令の定義例を示すブロック図である。

「複合命令」とは、2つの16ビット長の命令を組み合わせた命令である。図10において、左列および中央列のブロックは既存の16ビット長命令のニーモニック表記8であり、右列のブロックは既存の16ビット長命令を2つ組み合わせて2命令分の処理内容を表すニーモニック表記9である。図10に示すように、複合命令のニーモニック表記9は、2つの16ビット長命令のすべての組み合わせについて定義されている。

【0062】実施の形態3のアセンブラは、このような複合命令で記述されたソースファイルから命令コード（機械語）を生成するものであり、このためコード変換部904は、ソースファイルから読み込んだ複合命令を、コード変換テーブル910を参照して、対応する複合命令の命令コード（機械語）に変換するようになっている。コード変換テーブル910は、図10の定義例に示した複合命令のニーモニック表記9と複合命令の命令コードとを、前述した図14に示す形式でように対応づけたものである。

【0063】図11は、実施の形態3のアセンブラによるアセンブラ処理手順を示すフローチャートである。図11に示すように実施の形態3のアセンブラでは、まずバッファを初期化する（ステップS1101）。そして、ファイル入力部2によって、複合命令で記述されたソースファイルから複合命令を一行読み込み（ステップS1102）、読み込んだ複合命令を構文解析部3によって解析する（ステップS1103）。つぎに、コード変換部904によって、コード変換テーブル910を参照して複合命令のニーモニックを32ビット長の命令コード（機械語）に変換し（ステップS1104）、変換された機械語をバッファへ保存する（ステップS1105）。バッファに保存された機械語は、ファイル出力部7によって、エンディアンに従ってオブジェクトファイルに生成する（ステップS1106）。このようなステップS1102からステップS1106までの処理を、ソースファイルに記述されているすべての行の複合命令について繰り返す。

【0064】このように実施の形態3のアセンブラでは、図10に示すような複合命令を定義して、この複合命令のニーモニック表記と命令コードとを対応づけるコード変換テーブル910を使用してコード変換部904により命令コードを生成しているので、生成される命令コードは全て32ビット長となる。このため、プロセッサのリトルエンディアン動作における命令の逆転実行を確実になくすことができ、また生成される命令コードのコード量の最適化を図ることが可能となる。

【0065】

【発明の効果】以上説明したように、この発明によれば、プロセッサがビッグエンディアンとリトルエンディ

アンのどちらのエンディアンで動作する場合でも、アセンブリ言語記述の同一ソースファイルを用いて同一の処理を実行するオブジェクトファイルを生成することができ、ソフトウェアの生産性を改善することができるという効果を奏する。

【0066】つぎの発明によれば、全ての16ビット長命令を32ビットの境界に配置して、リトルエンディアン動作時において16ビット長命令がアドレスに逆行した実行動作となることを無視することができるという効果を奏する。また、逆アセンブルでメモリイメージとニーモニックの対応表示を行った場合にはコーディングした手順がそのまま出力されるため、検証には都合がよくソフトウェア開発の効率化を図れるという効果を奏する。

【0067】つぎの発明によれば、プロセッサのリトルエンディアン動作時における命令の逆転実行を確実になくすことができ、また、コード量の最適化を図れるという効果を奏する。

【0068】つぎの発明によれば、プロセッサがビッグエンディアンとリトルエンディアンのどちらのエンディアンで動作する場合でも、アセンブリ言語記述の同一ソースファイルを用いて同一の処理を実行するオブジェクトファイルを生成することができ、ソフトウェアの生産性を改善することができるという効果を奏する。

【0069】つぎの発明によれば、全ての16ビット長命令を32ビットの境界に配置して、リトルエンディアン動作時において16ビット長命令がアドレスに逆行した実行動作となることを無視することができるという効果を奏する。また、逆アセンブルでメモリイメージとニーモニックの対応表示を行った場合にはコーディングした手順がそのまま出力されるため、検証には都合がよくソフトウェア開発の効率化を図れるという効果を奏する。

【0070】つぎの発明によれば、プロセッサのリトルエンディアン動作時における命令の逆転実行を確実になくすことができ、また、コード量の最適化を図れるという効果を奏する。

【0071】つぎの発明によれば、上記の発明のいずれか一つの動作をコンピュータによって実行することができるという効果を奏する。

【図面の簡単な説明】

【図1】 この発明の実施の形態1および実施の形態2のアセンブラの構成を示すブロック図である。

【図2】 図1に示した実施の形態1のアセンブラによるビッグエンディアン用命令コード生成処理手順を示すフローチャートである。

【図3】 図1に示した実施の形態1のアセンブラによるリトルエンディアン用命令コード生成処理手順を示すフローチャートである。

【図4】 図4(a)は、図1に示した実施の形態1の

アセンブラにおいて、ビッグエンディアン用命令コード生成時の32ビット長の命令コードA、16ビット長の命令コードB、および16ビット長の命令コードCの一時保管バッファへの格納状態、およびオブジェクトファイルへの出力状態を示す説明図である。図4(b)は、図1に示した実施の形態1のアセンブラにおいて、ビッグエンディアン用命令コード生成時の16ビット長の命令コードB、32ビット長の命令コードAの一時保管バッファへの格納状態、およびオブジェクトファイルへの出力状態を示す説明図である。

【図5】 図5(a)は、図1に示した実施の形態1のアセンブラにおいて、リトルエンディアン用命令コード生成時の32ビット長の命令コードA、16ビット長の命令コードB、および16ビット長の命令コードCの一時保管バッファへの格納状態、およびオブジェクトファイルへの出力状態を示す説明図である。図5(b)は、図1に示した実施の形態1のアセンブラにおいて、リトルエンディアン用命令コード生成時の16ビット長の命令コードB、32ビット長の命令コードAの一時保管バッファへの格納状態、およびオブジェクトファイルへの出力状態を示す説明図である。

【図6】 この発明の実施の形態2のアセンブラの構成を示すブロック図である。

【図7】 図6に示したアセンブラによるビッグエンディアン用命令コード生成処理手順を示すフローチャートである。

【図8】 図6に示した実施の形態2のアセンブラによるリトルエンディアン用命令コード生成処理手順を示すフローチャートである。

【図9】 この発明の実施の形態3のアセンブラの構成を示すブロック図である。

【図10】 図9に示した実施の形態3のアセンブラにおいて使用される複合命令の定義例を示すブロック図である。

10

20

*

*【図11】 図9に示した実施の形態3のアセンブラによるアセンブラ処理手順のフローチャートである。

【図12】 図1、図6および図9に示したアセンブラを適用するプロセッサの命令の種類を示す図である。

【図13】 図1、図6および図9に示したアセンブラを適用するプロセッサの命令実行手順を示すフローチャートである。

【図14】 図1および図6に示したアセンブラのコード変換部で参照するコード変換テーブルの説明図である。

【図15】 機械語に変換した16進数コードのメモリ配置を示す図である。

【図16】 ビッグエンディアンにおける機械語のメモリ配置を示す説明図である。

【図17】 リトルエンディアンにおける機械語のメモリ配置を示す説明図である。

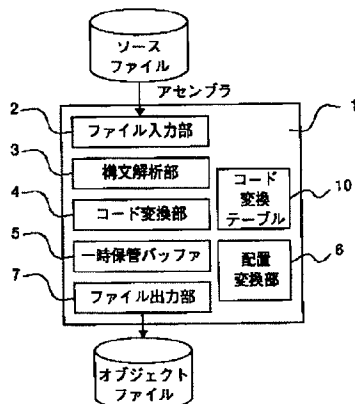
【図18】 従来のアセンブラの構成を示すブロック図である。

【図19】 従来のアセンブラにおけるアセンブラ処理手順のフローチャートである。

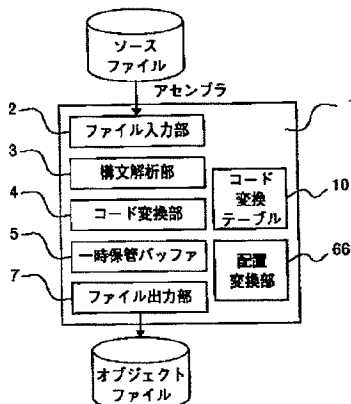
【符号の説明】

1 アセンブラ、2 ファイル入力部、3 構文解析部、4 コード変換部、5 一時保管バッファ、6 配置変換部、7 ファイル出力部、8 ニーモニック表記、9 複合命令のニーモニック表記、10 コード変換テーブル、11 アセンブラ、12 ファイル入力部、13 構文解析部、14 コード変換部、17 ファイル出力部、18 ニーモニック、19 機械語、20 ニーモニック、21 機械語(命令コード)、22 アドレス、23 命令コードの配置(ビッグエンディアン)、24 命令コードの配置(リトルエンディアン)、66 配置変換部、904 コード変換部、910 コード変換テーブル。

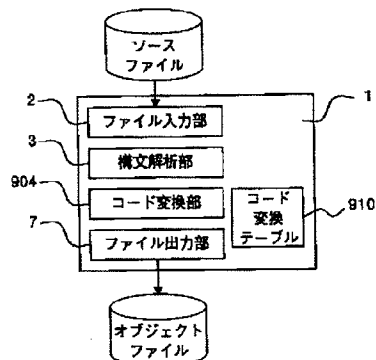
【図1】



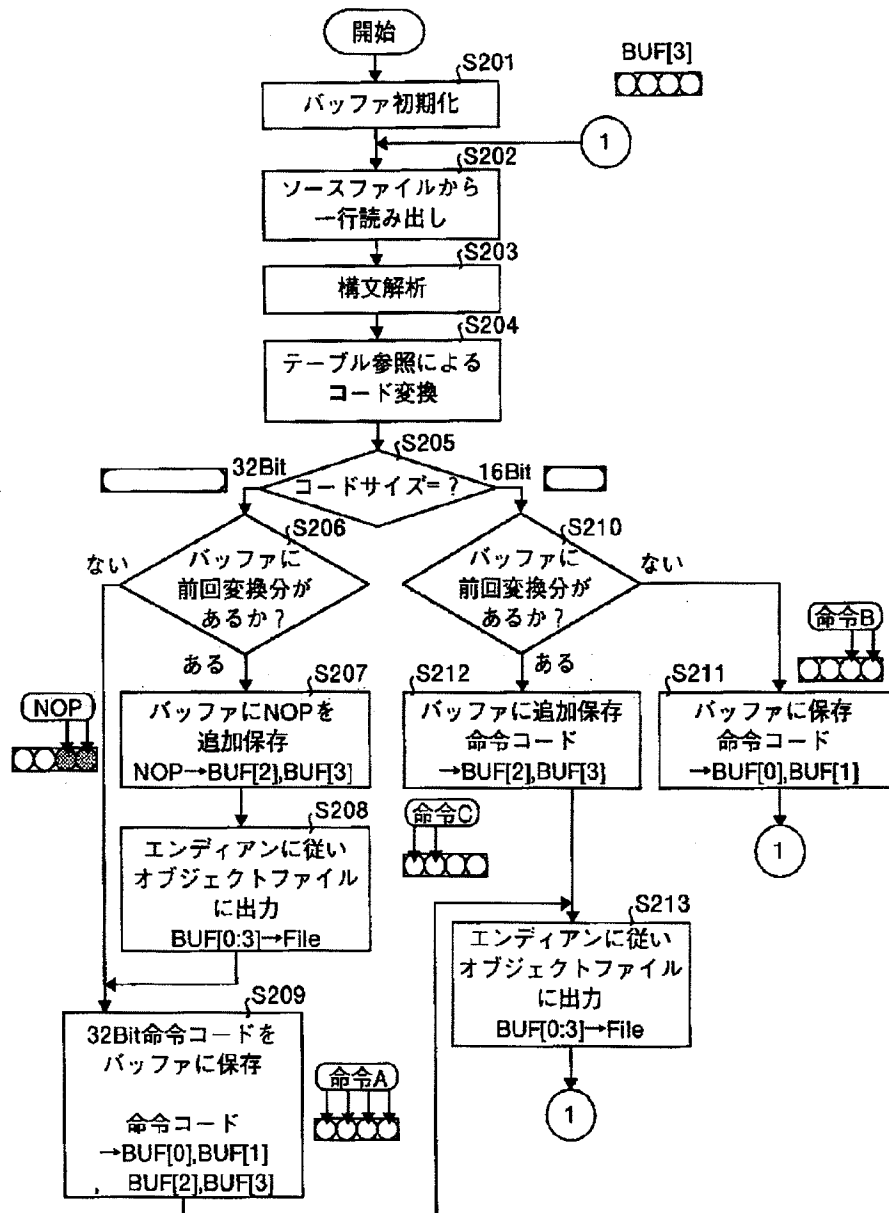
【図6】



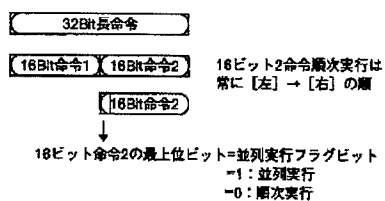
【図9】



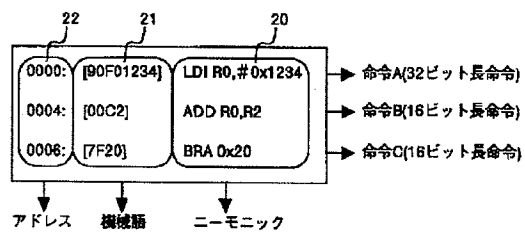
【図 2】



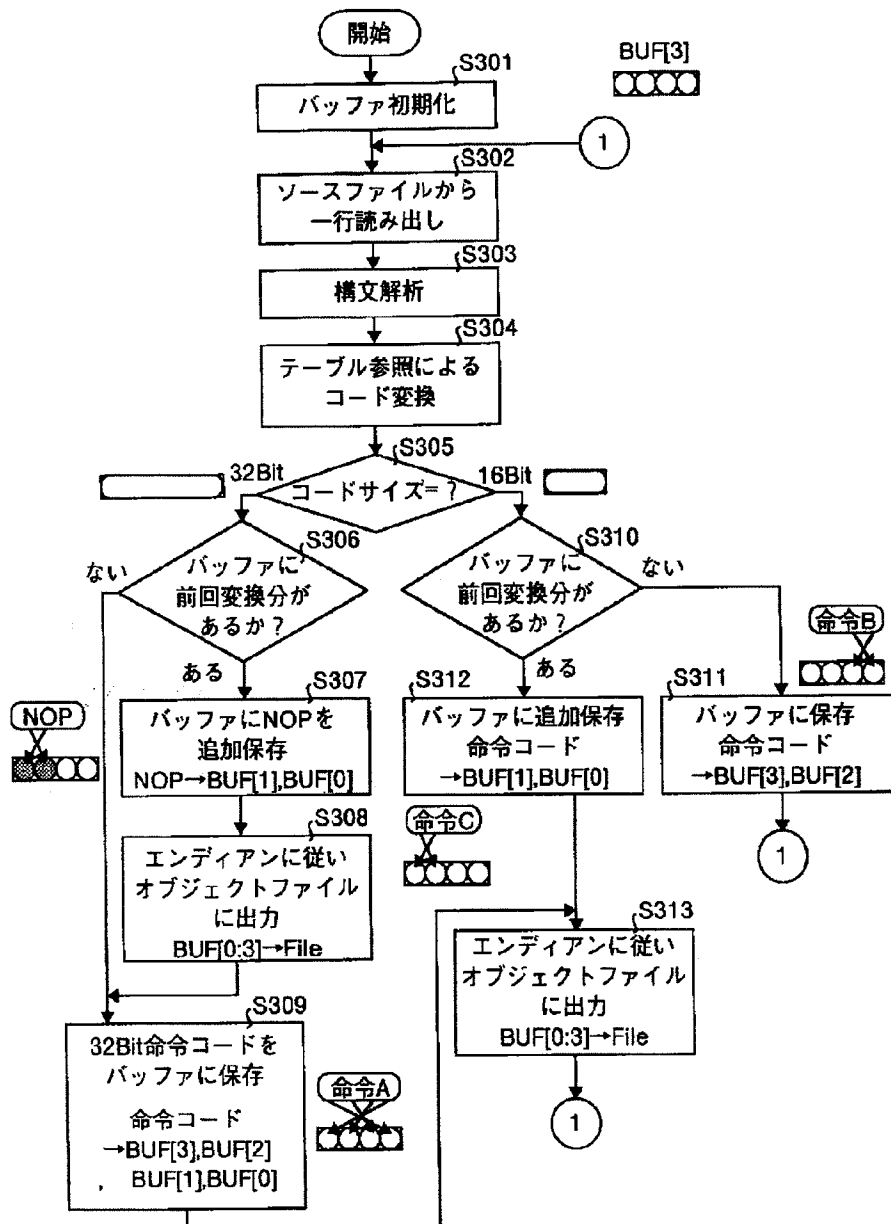
【圖 12】



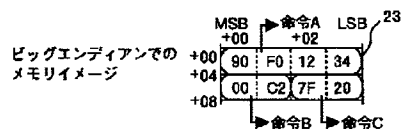
【図 15】



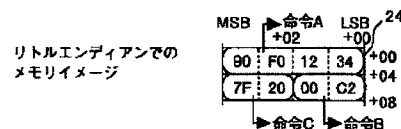
【図3】



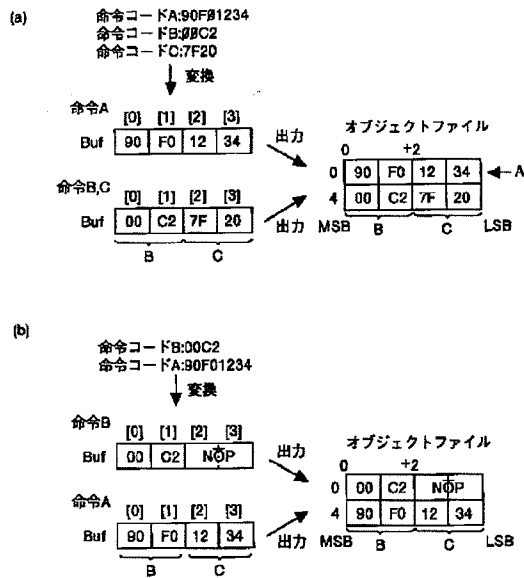
【図16】



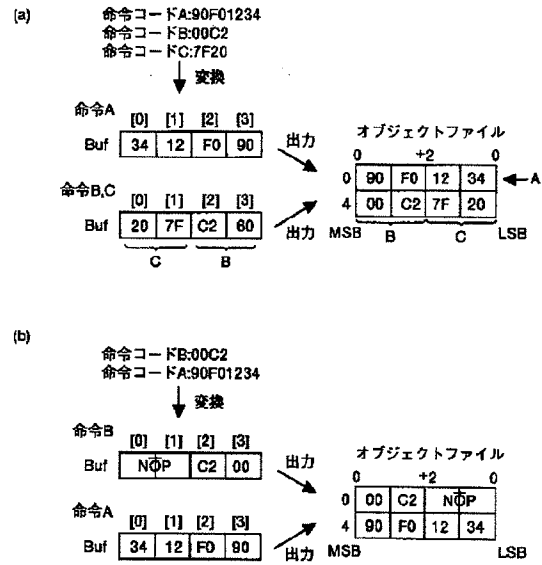
【図17】



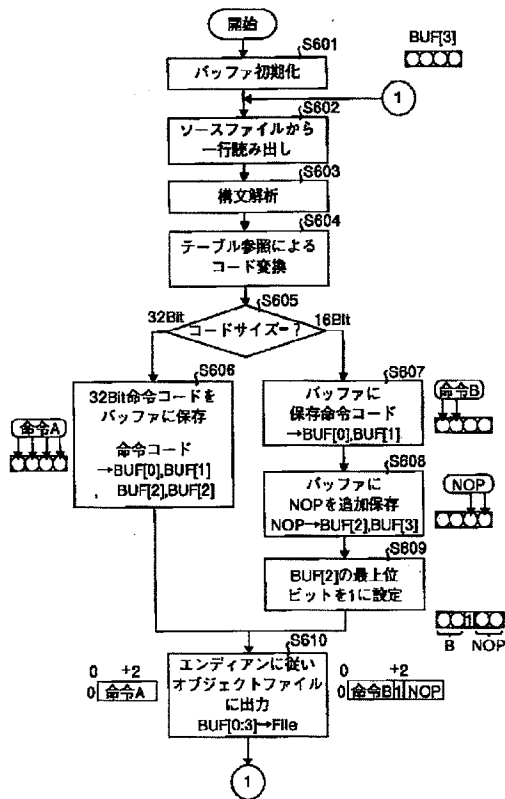
【図4】



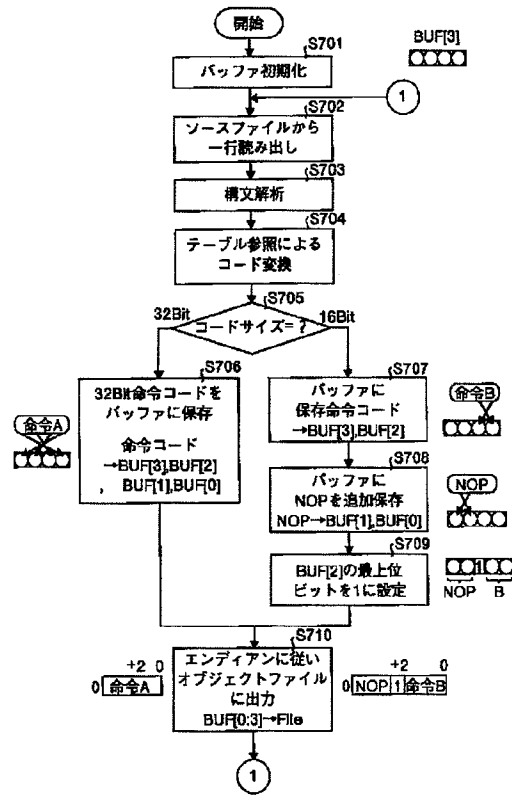
【図5】



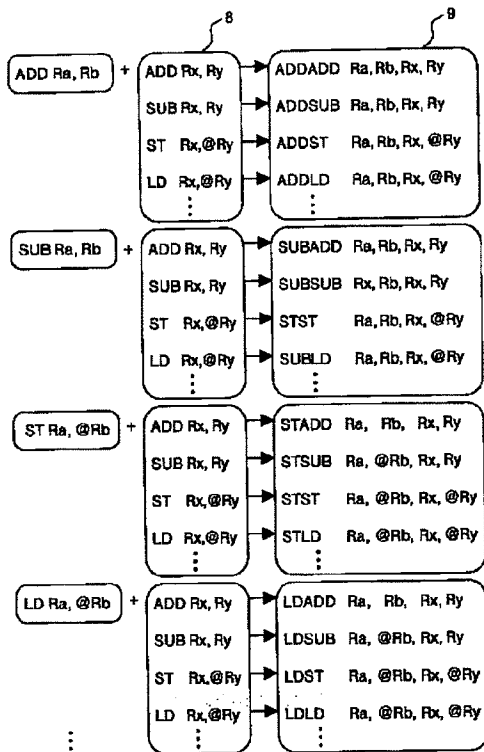
【図7】



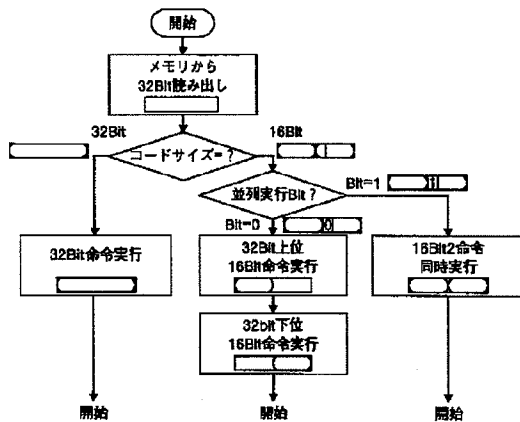
【図8】



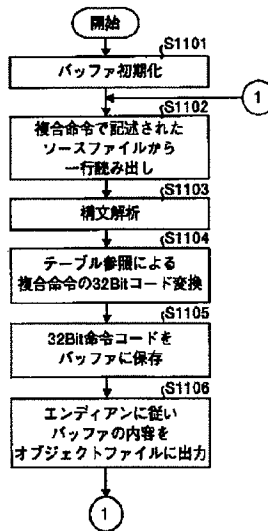
【図10】



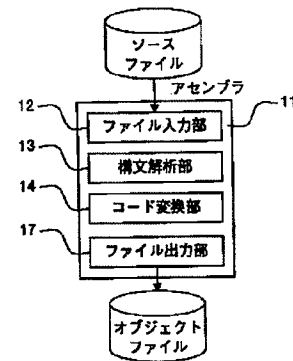
【図13】



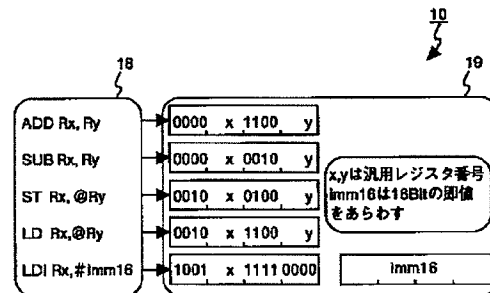
【図11】



【図18】



【図14】



【図19】

